Real-Time Fusion of Image and Inertial Sensors for Navigation

J. Fletcher, M. Veth, and J. Raquet
Air Force Institute of Technology

BIOGRAPHY

Jordan Fletcher is a Communications and Electrical Engineering Officer in the United States Air Force. He holds a B.S. in Computer Engineering from Tulane University. He is currently a Computer Engineering Master's student at the Air Force Institute of Technology, with a focus on navigation technology and exploitation of graphics hardware for general-purpose computing.

Major Mike Veth is an Assistant Professor in the Department of Electrical and Computer Engineering at the Air Force Institute of Technology. His current research focus is on the fusion of optical and inertial systems. He received his Ph.D. in Electrical Engineering from the Air Force Institute of Technology, and a B.S. in Electrical Engineering from Purdue University.

John Raquet is an Associate Professor in the Department of Electrical and Computer Engineering at the Air Force Institute of Technology, where he is also the Director of the Advanced Navigation Technology (ANT) Center. He has been involved in navigation-related research for over 17 years.

ABSTRACT

As evidenced by many biological systems, the fusion of optical and inertial sensors represents an attractive method for passive navigation. In our previous work, a rigorous theory for optical and inertial fusion was developed for precision navigation applications. The theory was based on a statistical transformation of the feature space based on inertial sensor measurements. The transformation effectively constrained the feature correspondence search to a given level of a priori statistical uncertainty. When integrated into a navigation system, the fused system demonstrated performance in indoor environments which were comparable to that of GPS-aided systems.

In order to improve feature tracking performance, a robust feature transformation algorithm (Lowe's SIFT) was chosen. The SIFT features are ideal for navigation applications in that they are invariant to scale, rotation, and illumination. Unfortunately, there exists a correlation between feature complexity and computer processing time. This limits the effectiveness of robust feature extraction algorithms for real-time applications using traditional microprocessor architectures. While recent advances in computer technology have made image processing more commonplace, the amount of information that can be processed is still limited by the power and speed of the CPU. In this paper, a new theory which exploits the highly parallel nature of General Programmable Graphical Processing Units (GPGPU) is developed which supports deeply integrated optical and inertial sensors for real-time navigation.

Recent advances in GPGPU technology have made real-time, image-aided navigation a reality. Our approach leverages the existing OpenVIDIA core GPGPU library and commercially available computer hardware to solve the image and inertial fusion problem. The open-source libraries are extended to include the statistical feature projection and matching techniques developed in our previous research.

The performance of the new processing method was demonstrated by integrating the inertial and image sensors on a commercially-available laptop computer containing a programmable GPU. Experimental data collections have shown up to a 3000% improvement in feature processing speed over an equivalent CPU-based algorithm. In this experimental configuration, frame rates of greater than 10 Hz are demonstrated, which are suitable for real-time navigation. Finally, the navigation performance of the new real-time system is shown to be identical to that of the old method which required lengthy post-processing.

INTRODUCTION

The advent of the Global Positioning System (GPS) has revolutionized the navigation community, and as a result we are increasingly dependent on navigation systems, and we want to be able to navigate in all environments. While GPS works well in many environments, it is not ideal for navigation indoors, underground, in urban canyons, or in

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2007		**00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Real-Time Fusion of Image and Inertial Sensors for Navigation	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Air Force Institute of Technology,Department of Electrical and Computer Engineering,Wright Patterson AFB,OH,45433**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

As evidenced by many biological systems, the fusion of optical and inertial sensors represents an attractive method for passive navigation. In our previous work, a rigorous theory for optical and inertial fusion was developed for precision navigation applications. The theory was based on a statistical transformation of the feature space based on inertial sensor measurements. The transformation effectively constrained the feature correspondence search to a given level of a priori statistical uncertainty. When integrated into a navigation system, the fused system demonstrated performance in indoor environments which were comparable to that of GPS-aided systems. In order to improve feature tracking performance, a robust feature transformation algorithm (Lowe?s SIFT) was chosen. The SIFT features are ideal for navigation applications in that they are invariant to scale, rotation, and illumination. Unfortunately, there exists a correlation between feature complexity and computer processing time. This limits the effectiveness of robust feature extraction algorithms for real-time applications using traditional microprocessor architectures. While recent advances in computer technology have made image processing more commonplace, the amount of information that can be processed is still limited by the power and speed of the CPU. In this paper, a new theory which exploits the highly parallel nature of General Programmable Graphical Processing Units (GPGPU) is developed which supports deeply integrated optical and inertial sensors for real-time navigation. Recent advances in GPGPU technology have made realtime, image-aided navigation a reality. Our approach leverages the existing OpenVIDIA core GPGPU library and commercially available computer hardware to solve the image and inertial fusion problem. The open-source libraries are extended to include the statistical feature projection and matching techniques developed in our previous research. The performance of the new processing method was demonstrated by integrating the inertial and image sensors on a commercially-available laptop computer containing a programmable GPU. Experimental data collections have shown up to a 3000% improvement in feature processing speed over an equivalent CPU-based algorithm. In this experimental configuration, frame rates of greater than 10 Hz are demonstrated, which are suitable for real-time navigation. Finally, the navigation performance of the new real-time system is shown to be identical to that of the old method which required lengthy post-processing.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**11**	

the presence of intentional or unintentional radio frequency (RF) interference. Unfortunately, many existing non-GPS forms of navigation have problems in the same types of environments.

The navigation system presented in this paper is inspired by biological systems. Many animals have the innate ability to navigate naturally simply using biological sensors. Some animals, such as sea turtles, use the electro-magnetic fields of the Earth to navigate [13]. Ants and many other insects use chemical trails to follow a path [15]. More elaborate biological navigation systems use sonar sensors to get an estimate of terrain and distance [9]. However, all of these biological systems are susceptible to interference, jamming, and changing medium, and therefore exhibit the same weaknesses as GPS systems. Animals that use passive sensors, such as bees, migratory birds, and humans, however, are not susceptible to these problems. These animals all use inertial and imaging sensors to pick out landmarks in the environment and navigate from landmark to landmark with relative ease. In addition, these sensors are passive, self-contained, and do not have the same weaknesses as GPS.

When used separately, however, imaging and inertial sensors have drawbacks that can result in poor navigation performance. An inertial sensor relies upon dead-reckoning [1] for navigation, which is susceptible to drift over time. Imaging sensors can have difficulty in identifying and matching good landmarks for navigation [3]. To solve this problem, the sensor readings in the navigation system were fused at a low level and used to correct each other.

Previous image-aided inertial navigation systems have been a trade-off of power and performance. Some have used a simplified image processing algorithm [30], or *a priori* navigation information [3], while others have simply post-processed navigation data [31], [3]. These solutions are not robust enough for use in autonomous navigation systems or as a viable GPS alternative. The navigation system described in this paper achieves real-time performance using a complex image processing algorithm that can work in a wide variety of environments.

INERTIAL NAVIGATION

Inertial navigation systems (INS) exploit measurements of specific force and angular rate to estimate the position, velocity, and attitude of the device. For six degree of freedom navigation, triads of accelerometers and gyroscopes are usually employed. The inertial navigation system is inherently a dead-reckoning device as the navigation states are propagated by integration (cumulative summing) of sensor measurements. As with all dead-reckoning navigation systems, the navigation error is subject to unconstrained errors [1], which are collectively defined as the sensor drift.

The inertial navigation system errors can be estimated and corrected by incorporating additional measurements. For terrestrial applications, the zero-velocity update is easily implemented and, as such, quite commonly used in systems such as [31],[20]. Another common measurement source for estimating the navigation state errors is the Global Positioning System [22],[30]. Unfortunately, GPS measurements are not always available, which motivates the development of additional navigation state measurement sensors. Navigation aiding using imaging sensors is described in the next section.

IMAGE-AIDING

Image sensors can also be used to correct an INS. Most image-aided inertial systems work by first identifying features (landmarks, edges, corners) in an image. The process of extracting features from the actual sensor readings is known as feature extraction, or feature transformation, of the image. Image-matching algorithms match features from an image taken at one point in time to features from an image taken at a later point in time. The process of matching features between successive images is known as feature correspondence.

The simplest way to perform feature correspondence is to perform a search over the entire space of another image, also known as an exhaustive search. This procedure is obviously computation-intensive and costly, in terms of time and computation resources. For example, in a system with N_k features in image k, the complexity of performing an exhaustive feature correspondence search in Big-O notation [11] is $O(N_k * N_{k+1})$. This relationship is illustrated in Fig. 1.

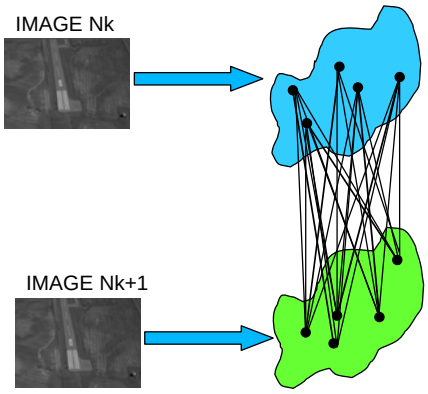

***Figure 1**: Exhaustive method of correspondence search. Given the projection of image N_k and successive image N_{k+1} into feature space, the exhaustive correspondence search is carried out by attempting to match every feature from N_k to every feature in N_{k+1}.*

However, when other sensors are available, it helps to use these sensors to constrain the correspondence search space. In an image-aided INS, inertial measurements can be used to predict feature locations from one image to the next. The search area can then be constrained to a subset of the features in a successive image that are found within a certain distance from the predicted location. The resulting complexity is O (N_k). In addition, the difference between the predicted feature location and the actual feature location can be used to correct IMU biases. The constrained search is shown in Fig. 2.

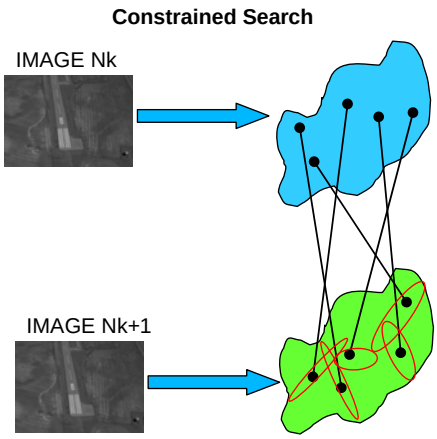

***Figure 2**: Constrained method of correspondence search. Given the projection of image N_k and successive image N_{k+1} into feature space, the constrained correspondence search is carried out by attempting to match every feature from N_k to an constrained area around the predicted location in N_{k+1}.*

Systems that use feature matching have been of limited value in the past, because most image processing algorithms were not suited to navigation purposes. Image processing for navigation requires features that can be uniquely identified despite movement of the sensor platform. The platform movement results in features being viewed from multiple angles, distances, and among other similar objects. To accomplish this feat, a feature has to have some kind of unique identifier that is invariant to changes in orientation and scale. False positive identification is especially detrimental in feedback-enabled navigation, since the landmark data is used to correct other system sensors. False identification will corrupt the other sensor readings and lead to poor navigation prediction. The main problem with using complex features, from a computation standpoint, is that feature extraction requires extensive computation [3]. This problem is compounded by the fact that the identifier needs to be sufficiently complex to ensure uniqueness, which increases feature correspondence search times.

To overcome this problem, past systems have used *a priori* environment information to choose features. One such system, MOB-LAB, navigated streets by using the lines on the road as features [2]. The problem with this approach is that the environment becomes un-navigable when lines are unavailable, such as when the vehicle goes off-road. Other systems, such as Minerva [29] and [21] tracked lights and ceiling tiles to determine their position at any epoch in time. Algorithms that use *a priori* information in this manner can be classified into a subset of navigation systems that rely upon reference landmarks for navigation. Such systems often use landmarks with a known, fixed coordinate (i.e., waypoints) to help guide and correct the navigation system. These systems have been met with some recent success, as Stanford has found with their "Stanley" vehicle for the DARPA Grand Challenge, 2005 [26]. However, to make an image-aided navigation system work in unknown environments, it must choose features that are invariant to scale and rotation without any *a priori* feature information.

SENSOR INTEGRATION

The previous two sections described how position can be accurately determined using inertial sensor readings for dead-reckoning navigation and feature correspondence from image sensors to correct the IMU estimates. However, the IMU estimates are in three-dimensional coordinates, while the image features are found in a two-dimensional Cartesian plane. To get accurate 3D position for a feature, the 2D coordinate must be augmented to a 3D coordinate, which requires additional information, namely, the line-of-sight distance from the navigating platform. This is impossible in most situations using an image sensor alone. *A priori* terrain maps are necessary to extract distance information using a monocular image sensor. However, if the navigation platform has multiple image sensors (e.g., binocular vision), the distance to a feature can be calculated using information about the image sensors, their relation to each other, and their relation to the navigation platform [31]. Another way to obtain distance information is to use a single image sensor and a ranging sensor, such as LIDAR [27] or sonar [4].

Obviously, better results can be obtained using additional sensors. To optimally use these additional sensors, the theory of multi-sensor fusion must be taken into account. This way the system can perform optimally given any number of sensors without redesigning the underlying system architecture. The navigation system in this paper actively acquires sensor readings and fuses the data into a single, combined navigation estimate. Therefore, the delay in navigation prediction is directly tied to two factors: the time it takes to acquire sensor readings and the time it takes to fuse the data.

The navigation system presented in this paper ensures optimal sensor data acquisition by decoupling the sensor activity from the navigation state prediction. Decoupling is accomplished by using a thread-safe, centrally-shared

data storage object between all the sensors and the navigation component of the software. Each sensor has the ability to write to a particular area of shared memory (section of the blackboard) and the navigation component has access to all the sensor data. Thread synchronization for navigation reading and sensor writing is accomplished via use of mutually exclusive locks and conditions. The blackboard architecture is not new. It was developed by Artificial Intelligence researchers over a decade ago, and is described in detail in [8] and [5]. This architecture applied to sensor fusion was met with success in the 2005 DARPA Grand Challenge [24]. The strength of this technique is that as more sensors are added to the platform, navigation predictions are not slowed down while waiting for sensor acquisition. This makes the approach flexible and extensible for additional sensors. The blackboard architecture for a multi-agent (multi-sensor/multi-thread) software platform is illustrated in Fig. 3.

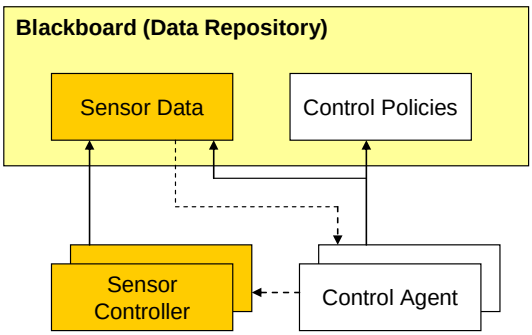

***Figure 3**: Blackboard architecture. This software architecture allows multiple sensors to acquire data and write to a central, shared data repository. Control agents can then access the shared sensor data and write policies that govern access to this data by multiple agents.*

The navigation component of the navigation system is responsible for fusing the acquired data that has been written to the blackboard into a single navigation prediction at any epoch in time. This fusion is currently performed using an extended Kalman filter. This fusion method was chosen because of its successful application in previous work [24], [21]. The conceptual design of the Kalman filter applied to multi-sensor fusion is shown in Fig. 4.

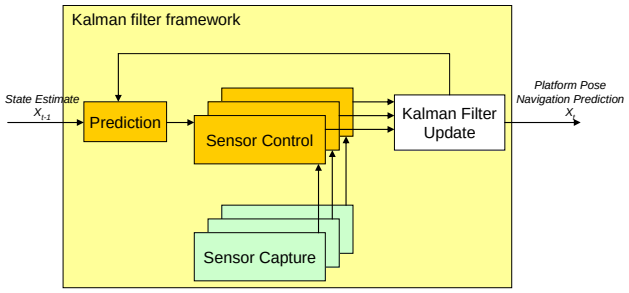

***Figure 4**: Conceptual Kalman filter for a multi-sensor system. The previous state $\boldsymbol{X}_{t-1}$ is used to create a prediction for the next state, $\boldsymbol{X}_t$. Multiple sensors are used as additional input to the prediction and combined in the Kalman filter to come up with the navigation prediction. In addition, the Kalman filter can be used in a closed-feedback system to correct future predictions.*

The Kalman filter is an optimal, recursive, data processing algorithm used to estimate the state (position, heading and orientation) of a dynamic system from incomplete, noisy measurements (sensor readings). To predict the current navigation state at any time, *t*, the state $\boldsymbol{X}_t$ is computed using the previous state $\boldsymbol{X}_{t-1}$, and inputs from the system sensors. In addition, the Kalman filter can compute the estimated accuracy of the state estimate. This accuracy is computed in the error covariance matrix, $\boldsymbol{P}_t$.

The second phase of a Kalman filter is the update phase, which uses measurement information from the current time to refine the prediction and come up with better estimates. For this application, the measurement information is the residual results from feature correspondence. The residual distance between feature predictions and actual locations is used to adjust the biases for the inertial sensors. The multi-sensor Kalman filter, adjusted for imaging and inertial fusion, is shown in Fig. 5. For more information on Kalman filtering, refer to [21], [32] and [16].

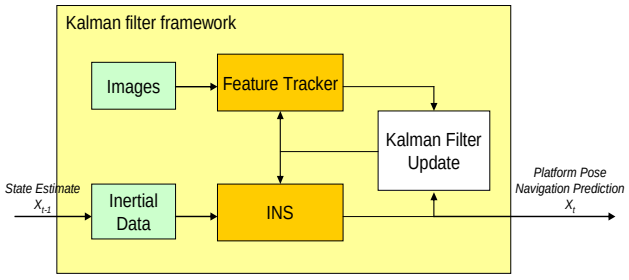

***Figure 5**: Kalman filter used for the image-aided inertial navigation system. Images and Inertial Data are the sensor readings, which are used as inputs for the navigation predictions by the Feature Tracker and INS. The Kalman filter combines the predictions and performs feedback to correct the INS and Feature Tracker module. In this way, the Kalman filter removes errors from the INS trajectory.*

REAL-TIME PROCESSING

Real-time processing has been redefined many times over the past few decades. The term real-time originally referred to any system that could perform computations at least as fast as the real process it was simulating [12]. This definition changed during the advent of thread priority scheduling and programmable microcontrollers. These technologies allowed a process to pre-empt the operating system, resulting in increased performance, reliability and predictability.

Current real-time systems are now divided into two categories; hard real-time systems and soft real-time systems. Both categories are governed by the idea of a deadline from an event to a system response. The two differ in that a hard real-time system considers the system response after a deadline to be useless, whereas the soft real-time system will tolerate a missed deadline (usually with some impact on performance), and continue to operate [12]. Hard real-time systems are thus best suited to safety-of-life applications, while soft real-time systems are suited to applications that involve concurrent access with changing situations.

The navigation system presented in this paper relies upon concurrent access to data from multiple sensors, and is therefore best suited for a soft real-time system. The event associated with this real time system is sensor data acquisition, and the response is a navigation prediction. The deadline can vary depending on the application, but is limited by the maximum capture rate of the slowest sensors. For example, a typical image sensor can capture an 800x600 monochrome image at 30 frames per second (FPS), or 30 Hz. A typical IMU can capture inertial data at rates exceeding 100 Hz. Therefore, the deadline should be set to 33 ms or greater, since that is the capture rate of the slowest sensor.

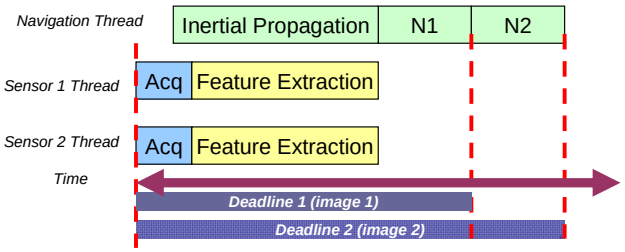

***Figure 6**: Deadline timing for 2 image sensor threads. Sensor 1 and sensor 2 simultaneously acquire images and perform parallel feature extraction on the GPU. N1 and N2 represent the navigation calculations for sensor 1 and sensor 2, respectively, which are performed in serial. The deadlines are the elapsed time between the initial image capture time and the time when the navigation prediction is made available.*

A dynamic deadline is utilized in this navigation system. Features are extracted from the image sensors as the images are acquired, and written to the data repository. The navigation component runs concurrently in a separate thread and accesses the sensor data from the data repository as it becomes available. It uses this data to perform inertial propagation, a feature correspondence search, and combines all the data in a Kalman filter. The total time between sensor acquisition and navigation state computation (deadline) is the sum of the time between the image capture signal and the navigation component computations. The diagram in Fig. 6 illustrates this relationship.

Processing navigation predictions concurrently with sensor data acquisition is much more complicated than a serial procedure. The two processes need to be synchronized and controls need to be in place to ensure that sensor data is not acquired faster than it can be used. Data acquisition at a faster rate than data processing would overrun the data repository after a length of time. Synchronization for data structure reads and writes is achieved via mutually-exclusive locks and condition objects. For this navigation system, navigation state computation is kept relatively simple to ensure that it runs at a much faster rate than sensor acquisition. In addition, the computationally-intensive image processing step is coupled to the image sensor data acquisition, which adds an additional constraint and limits the sensor rate to one that is acceptable for processing. The only problem with coupling image processing to the data acquisition step is that the image capture rate is limited by the speed of feature extraction.

SIFT FEATURE EXTRACTION

The feature extraction algorithm used for the navigation system presented in this paper is called the Scale Invariant Feature Transform, or SIFT [14]. Conceptually, SIFT works by first decomposing an image into a scale-space representation. Decomposition results in a different type of information being available at each level of decomposition, but also allows the algorithm to separate extrema (local minima and maxima) in an image into the scale at which they are most prominent. Extrema are chosen from a progressively-scaled set of images that have been reduced to a lower level of detail via Gaussian blurring. Each resulting extremum is then assigned an orientation by taking a histogram of the gradient in a fixed area around it. The gradient around the extremum is used to compute a distinctive descriptor that can be used for matching features.

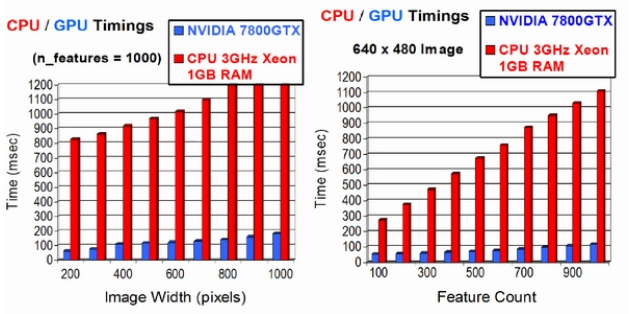

***Figure 7**: CPU vs. GPU timings for SIFT extraction for a GPU-based feature extraction program. For a 5x increase in width, the GPU experiences a 100 ms increase, while the CPU experiences over 400 ms increase in processing time.*

SIFT feature calculation is a time-consuming and computation-intensive process. The table in Fig. 7 summarizes the expected time to calculate SIFT features at various image resolutions and feature counts using a CPU and a GPU from [25].

The computation times for the CPU are not acceptable for real-time image-aided navigation. The image measurements are used in the update phase of the Kalman filter to correct dead-reckoning measurements. In between image measurements, the inertial data alone is used to determine the navigation prediction. If the system uses a consumer-grade IMU, then the navigation solution can drift significantly between image measurements. If the image measurements are far apart (> 1 second), the navigation solution will limited by the accuracy of the IMU sensor. The data set above shows that feature extraction alone on a large image (1280x1024) would take over 1 second, which is unacceptable for an accurate navigation solution.

Since sensor data acquisition is coupled to the speed of feature extraction, it is clear that the biggest navigation system gains come from faster image processing and feature extraction. This enables more image updates, resulting in more error corrections on the inertial measurements.

IMAGE PROCESSING ACCELERATION

The navigation system presented in this paper leverages the power of the programmable graphics processing unit to speed up feature extraction. Much like the programmable microprocessor was a leap forward in real-time system programming; the programmable GPU is a leap forward in high performance computing using commodity hardware. Typical speedups vary depending on implementation, but simple math operations performed on a GPU have found speedups between 30% and 400% over the CPU performance. The speedup is due to both the hardware configuration and processor architecture.

Modern video cards contain an onboard GPU dedicated to processing a limited number of operations on very large amounts of data. By contrast, the CPU is designed to perform a large number of operations on a small amount of data. This difference in purpose has led to divergent paths for CPU and GPU architecture. A typical GPU will have several processing pipelines working in parallel with each other, while a single-core CPU has a single processing pipeline.

Recent advances in multi-core CPU technology have made them more similar to the GPU in terms of architecture and performance. However, the processing power of the GPU is growing at a rate faster than Moore's Law [17], which models the CPU performance growth. The processing power gap is evident even when comparing older GPUs to single-core CPUs. Leading edge consumer-grade GPUs doubled the performance of leading-edge CPUs [10]. The table in Fig. 8 summarizes the processing power of various newer GPU architectures vs. CPU architectures using the Linpack Giga-flops (GFLOPS) rating [6].

GPU/CPU/Supercomputer	GFLOPS
Intel Pentium 4	10
NV 5900	20
Intel Dual Core	38
Intel Quad Core	62
NV 7800 GTX	200
NV 8800 GTX	330
Cray XT3 (10,000 CPU)	43,480
IBM BlueGene (100,000 CPU)	280,600

***Figure 8**: CPU vs. GPU vs. Supercomputer performance using the Linpack GFLOPS rating, a measure of billions of floating point operations possible per second. The supercomputers are suffixed by the number of processors they contain.*

The table makes one point very clear: consumer-grade CPUs are outperformed in raw computational power by the specialized GPUs, and GPUs are only getting faster. One can make the case that GPUs are outmatched by the supercomputers on the list. However, the supercomputers run thousands of processors in parallel to achieve their GFLOPS rating. If they were scaled to a single processor, their rating would be one-hundredth that of a single GPU. The reason for the large disparity is the specialized architecture of the GPU versus the generalized architecture of the CPU. The GPU was built to perform stream processing on large amounts of data.

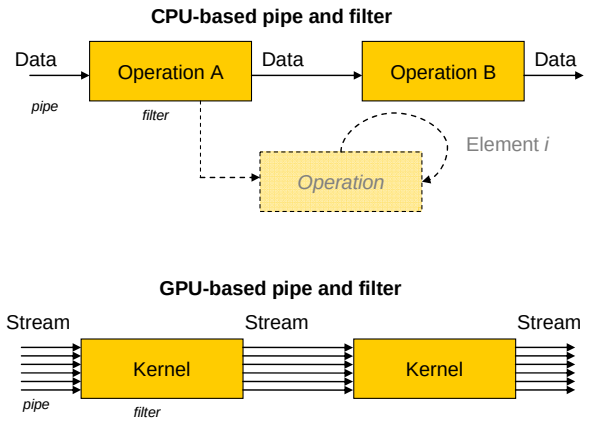

***Figure 9**: Pipe and filter on a CPU-based system vs. GPU. The CPU-based pipe and filter must perform serial operations on each element in the data stream, while a GPU can perform the same operation in parallel, using a kernel on parallel data sets.*

The stream-processing architecture also contributes to making the GPU well-suited for image processing applications. While a CPU is designed to operate on data serially, the GPU is designed for parallel data processing. This means that the larger the data set, the more advantage is gained from stream processing. One can view the difference as analogous to a pipe and filter architecture, where a CPU has a single pipe and multiple filters, whereas a GPU has multiple parallel pipes and multiple filters. The CPU must unroll the operations in each filter and perform the filter operations on each data element, while the GPU can perform the operation on multiple sets of data in parallel [28]. The two architectures are illustrated in Fig. 9.

GPU RENDERING PIPELINE

The GPU is designed to work on streams of vertex and fragment data. There are three basic stages to the rendering pipeline: vertex, texture, and fragment. The stages are diagrammed in Fig. 10 and explained in further detail in the following section.

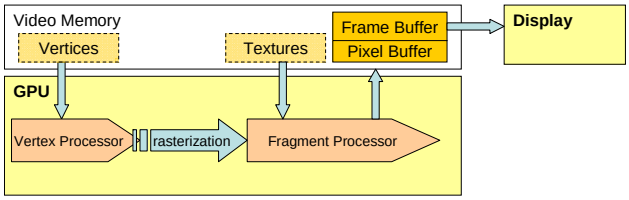

***Figure 10**: Programmable GPU rendering pipeline. Images and geometric primitives take different paths. Images are rendered by adding the image pixel values as a texture lookup, then rasterizing the entire image as geometry. Geometric primitives are drawn using a vertex processor, then applying texture lookups.*

Vertices describe 2D points in 3D space which make up primitive geometric shapes for a scene. The vertex data used to be static on non-programmable GPUs. To make changes to a geometric shape, the data had to be transferred to the CPU, altered, and then transferred back to the GPU. On modern programmable GPUs, vertices can be transformed in the GPU using custom vertex shader programs.

Textures are images that can be mapped onto surfaces of geometric primitives created in the vertex stage. In the texture stage of the rendering pipeline, the texture is referenced for the correct color to add to each fragment before they are converted to pixels.

A fragment is the term used to describe intermediate data in the processing pipeline before a pixel is drawn to the screen. Fragments can occupy more than one pixel location. Fragment shaders are customizable on modern GPUs and can also perform memory reads during texture lookups.

The last step of the graphics rendering pipeline is outputting pixels to the screen. This can be done either using a conceptual framebuffer object or using pixel buffer objects. The framebuffer is global "screen" memory. It can be corrupted if windows are stacked on top of each other or if windows are moved partially off-screen. This problem is avoided by using pixel buffer objects to store pixel data for off-screen rendering.

GENERAL-PURPOSE GPU

To utilize the graphics rendering pipeline for general-purpose computation, the data must be transformed to the native GPU format and architecture. The simplest way to do so is by mapping input data to fragment data. Custom vertex programs are written to change the geometry of the image as needed. Custom fragment programs can then operate on a per-pixel basis, using lookups from the custom textures to transform color information for each fragment. These vertex and fragment shader programs can be written using a variety of APIs and high-level shading languages. For more information on programmable shaders and OpenGL, see [23] and [18].

The image-processing component of the navigation system described in this paper was written using the OpenGL API and Nvidia's CG high-level shading language. CG compiles C/C++-like shading programs into assembly language that can be understood by the GPU. An existing GPU-based feature extraction algorithm, OpeNVIDIA [7], was used as the basis for the CG scripts and frame buffer objects. The OpeNVIDIA fragment shader programs take data acquired from the image sensors and perform the SIFT feature extraction algorithm (described in the SIFT section) as progressive

filters. The filters are used to choose the best prospective feature candidates and compute SIFT descriptors for each feature. To avoid unnecessary memory allocation and to keep feature processing on the GPU, feature information is stored in the high-performance texture memory that held the original image.

EXPERIMENTAL RESULTS

Experimental results from the navigation system have been very positive. Sensor decoupling and multi-threading alone provide a noticeable speedup in processing time. GPU-accelerated feature extraction speedups are orders of magnitude greater than the expected baseline results, and free tremendous amounts of CPU capacity for other tasks. Lastly, improvements to the algorithm and data structure reuse result in much more efficient processing and better performance characteristics. Results in each of these areas are described in the sessions that follow.

RESULTS: SENSOR DECOUPLING

The navigation system created in our previous work [31] was capable of post-processing navigation data and coming up with a navigation solution with near GPS-level accuracy. The system suffered from slow navigation prediction computation because it was a single-threaded, serial application. The new system is a multi-threaded, highly-parallel software platform, which matches well to multi-sensor fusion problems. Data acquisition for each sensor was assigned to a separate thread, which alleviated problems of sensor priority and availability.

Several simulations were run without using the GPU for feature extraction to test the effect of sensor decoupling on the overall system. The simulations were run on a set of time-tagged images and inertial data during an indoor data collection experiment at AFIT in August, 2006. The image, feature, and inertial data were all loaded from file on-the-fly, to emulate sensor readings. The results were impressive despite the fact that all data was read from file, which classically results in worse performance. For a simulation that lasted 277 seconds in "real-time", the post-processing solution took 677 seconds to complete and the decoupled system took 276 seconds to complete, which meets the classic definition of real-time. The completion time was a result of running the sensor "acquisition" (reading from a file) in serial. The simulation was also run using parallel sensor acquisition, resulting in an additional 20% speedup. Even without the parallel sensors, the new system shows a 2-3x speedup over the previous solution. In addition, the navigation prediction was within 2 meters of the actual (true) position. A similar speedup was found when running longer simulations as well.

RESULTS: GPU ACCELERATION

The initial results for GPU-accelerated feature extraction were also very impressive. The feature extraction tests were performed on indoor images used for navigation, using a C#-based SIFT feature extraction program for the CPU [19], versus OpeNVIDIA for the GPU-based feature extraction. The feature extraction step, typically the majority factor of a real-time deadline, was reduced from several seconds to milliseconds for high-resolution images.

Additional tests were run using the GPU-accelerated feature extraction for navigation predictions. The tests were set up using binocular image sensors and a consumer-grade IMU. Initial results found sustainable frame rates of 10 to 16 FPS, which is suitable for use in real-time navigation. The dynamic deadline for the experimental system was found to vary between 300 and 500 ms, indicating that approximately 0.5 seconds of lag could be expected before the sensor platform would have reliable navigation data for any epoch in time. (The impact of this delay could be negated by using the INS to propagate forward in time while the calculation is ongoing). Navigation results from initial tests lacked accuracy due to poor sensor calibration. This problem is currently being addressed and should be rectified soon. This problem illustrates the importance of sensor calibration for an accurate navigation solution.

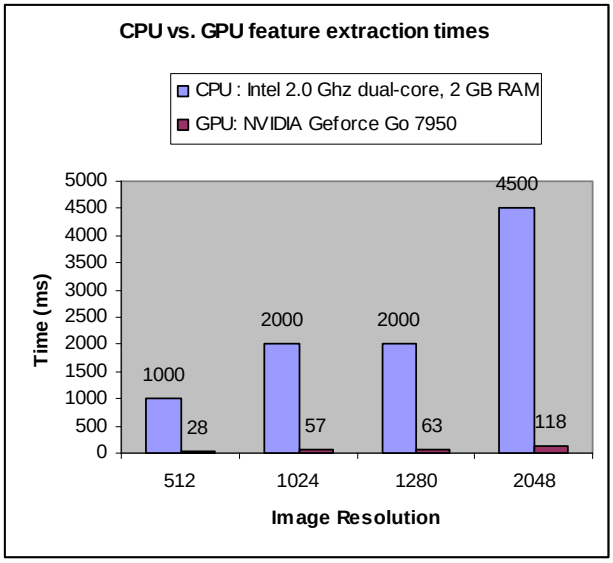

***Figure 11**: CPU vs. GPU feature extraction times using OpeNVIDIA on indoor images used for navigation. Although processing times generally increase due to image size, CPU times show a much larger increase due to image size than the GPU.*

The additional benefit from performing feature extraction on the GPU is that the CPU is freed for other tasks, as can be seen in Fig. 12. The majority of the CPU usage is due

to navigation prediction calculations. Further optimizations could be made by offloading more navigation computations to the GPU, which would free more CPU resources. In addition, many of the Kalman Filter calculations could be done in parallel, which the GPU can perform very efficiently.

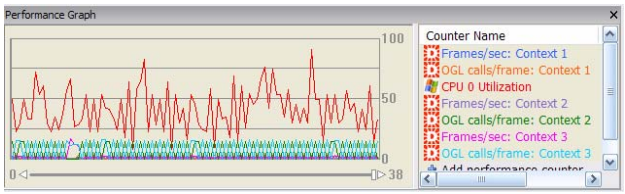

***Figure 12**: CPU and GPU performance graph running the navigation system. The graph was created using the gDebug tool from Nvidia. CPU usage is approximately 50% on average, GPU usage is approximately 20% on average.*

CONCLUSION

The navigation system presented in this paper represents a new area to explore in precision non-GPS navigation. The system is biologically-inspired, works in real-time, and fuses image-aided and inertial sensors. Two main strengths of this approach are the use of passive sensors for navigation and no requirements for *a priori* environment information. These strengths enable the navigation system to work in a variety of environments where previous navigations systems would fail.

The navigation system was designed with goals of near-optimal multi-sensor fusion and real-time processing. Both were achieved by building on the blackboard architecture style. This architecture allows the sensors to be decoupled from the navigation component, while simultaneously providing the navigation component access to all the sensor data. The sensors are fused through the use of an extended Kalman filter, which has been used successfully by many researchers in the past for similar problems.

The navigation system also harnesses the power of the Graphics Processing Unit to perform feature extraction on data from the image sensor. The feature extraction algorithm used is the Scale Invariant Transform, or SIFT, which has been an algorithm of choice for many navigation applications. SIFT's main drawback is the computation required to extract features, which has previously kept it from being used for real-time applications. This problem is overcome through the use of the GPU for general-purpose computation. Initial results have found a 4-30x speedup in feature extraction, which is a drastic improvement over equivalent mathematical algorithm speedups due to GPU processing. These speedups over the expected results are due to efficient stream processing and the PCI express interface, which allows a much higher communications bandwidth from the CPU to GPU.

Experimental results have seen a drastic improvement in processing speed for navigation simulations. The speedup achieved from decoupling the sensors and multi-threading doubles the speed of the previous work. The overall performance improvement has been approximately 10x–20x in conjunction with the GPU-accelerated feature processing. When taken in the context of live camera processing, the resulting navigation system can expect to run at several more frames per second, which results in a much more accurate navigation prediction.

The navigation system presented in this paper has many interesting implications. First, it runs real-time on mobile, commodity hardware. This means that the navigation system can be easily duplicated and mounted onto mobile platforms. Second, it runs in a variety of environments without the aid of GPS, waypoints or a priori information, making it well-suited to unknown, GPS-denied/unavailable environments. Third, it does not consume as many CPU resources as the previous image-aided inertial navigation system did. By offloading the majority of the computation to the GPU, resources are freed to run additional applications, such as an artificial intelligence, networking, data storage, or map-building applications.

DISCLAIMER

The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

ACKNOWLEDGEMENTS

The authors would like to thank the Air Force Research Laboratory for their sponsorship and funding of this research.

REFERENCES

[1] Nathaniel Bowditch. The American Practical Navigator, an Epitome of Navigation. National Imagery and Mapping Association, 1995.

[2] Alberto Broggi, Simona Berte. Vision-Based Road Detection in Automotive Systems: A Real-Time Expectation-Driven Approach. *Journal of Artificial Intelligence Research 3*, 1995.

[3] Zhenhe Chen, Jagath Samarabandu, Rango Rodrigo. Recent Advances in Simultaneous Localization and Map-building Using Computer Vision. University of Western Ontario Publishing, May 2007.

[4] Albert Diosi, Lindsay Kleeman. Advanced Sonar and Laser Range Finder Fusion for Simultaneous Localization and Mapping. Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems. Sendai, Japan, 2004.

[5] Jing Dong, Shanguo Chen, Jun-Jang Jeng. Event-based Blackboard Architecture for Multi-Agent Systems. The Proceedings of the IEEE International Conference on Information Technology Coding and Computing (ITCC), pages 379-384, April 2005.

[6] Jack Dongarra, Piotr Luszczek, Antoine Peitet. The LINPACK Benchmark: Past, Present, and Future. University of Tennessee, Computer Science Technical Report Number CS - 89 – 85, 2001.

[7] James Fung, Steve Mann, Chris Aimone. OpenVIDIA: Parallel GPU Computer Vision. *MULTIMEDIA*, pp 849-852. ACM Press, 2005.

[8] David Garlan, Mary Shaw. An Introduction to Software Architecture. Advances in Software Engineering and Knowledge Engineering, vol I. World Scientific Publishing Company, 1993.

[9] Donald Griffin. Bat Sonar. *Time Magazine*. May, 1950.

[10] Francis Kelly, Anil Kokaram. General Purpose Graphics Hardware for Accelerating Motion Estimation. Irish Machine Vision and Image Processing Conference (IMVIP), Sept 2003.

[11] Donald E. Knuth. Big Omicron and Big Omega and Big Theta. *SIGACT News*, 8(2):18-24, 1976.

[12] H. Kopetz. Real-Time Systems, Design Principles for Distributed Embedded Applications, Chpt. 10-11. Klower Academic Publishers, 1997.

[13] Kenneth Lohmann. Regional Magnetic Fields as Navigational Markers for Sea Turtles. *Science*, 2001.

[14] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[15] Lino Marques, Urbano Nunes, Aníbal T. de Almeida. Thin Solid Films. Volume 418, Issue 1, Oct 2002.

[16] Peter S. Maybeck. Stochastic Models Estimation and Control, Vol II. , 1979.

[17] Gordon Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 1965.

[18] Jackie Neider, Tom Davis, Mason Woo. The OpenGL Programming Guide (Red Book). Addison-Wesley Publishing Company, 1994.

[19] Sebastian Nowozin. Auto-pano SIFT. http://user.cs.tu-berlin.de/~nowozin/autopano-sift/.

[20] S. Panzieri, F. Pascucci, G. Ulivi. An Outdoor Navigation System Using GPS and Inertial Platform. IEEE/ASME Trans. on Mechatronics, vol. 7, n. 2, pp. 134-142, 2002, IEEE, USA.

[21] S. Panzieri, F. Pascucci, G. Ulivi. Vision based navigation using Kalman approach for SLAM. 11th Int. Conf. on Advanced Robotics, Portugal, 2003.

[22] L. Pinto, G. Forlani, D. Passoni. Experimental Tests on the Benefits of a More Rigorous Model in IMU/GPS System Calibration. National Research Program COFIN, Ministry of the University and Scientific Research of Italy, 2002.

[23] Randi Rost. The OpenGL Shading Language. Addison Wesley Publishing Company, 2006.

[24] G. Seetharaman, A. Lakhotia, et al. Technical Overview of CajunBot (2005). Technical Report for DARPA Grand Challenge, 2005.

[25] Sudipta N. Sinha, Jan-Michael Frahm, Marc Pollefeys, Yakup Genc. GPU-based Video Feature Tracking and Matching. EDGE 2006, workshop on Edge Computing Using New Commodity Architectures, Chapel Hill, May 2006.

[26] David Stavens, Hendrik Dahlkamp, Adrian Kaehler, Sebastian Thrun, Gary Bradski. Self-Supervised Monocular Road Detection in Desert Terrain. Technical Report for the 2005 DARPA Grand Challenge. 2006.

[27] Hartmut Surmann, Andreas Nuchter, Joachin Hertzberg. An Autonomous Mobile Robot with a 3D Laser Range Finder for 3D Exploration and Digitalization of Indoor Environments. Robotics and Autonomous Systems 45, pp 181-198, 2003.

[28] Chris J. Thompson, Sahngyun Hahn, Mark Oskin. Using Modern Graphics Architectures for General-Purpose Computing: A Framework and Analysis. In 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-35), 2002.

[29] S. Thrun, M. Beetz. Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva. *Journal of Robotics Research*, July 2000.

[30] S. Thrun, Daphne Koller, Zoubin Ghahramani, Hugh Durrant-Whyte, Andrew Y. Ng. Simultaneous Mapping and Localization with Sparse Extended Information Filters. *Journal of Robotics Research*, 2004.

[31] Michael Veth. Stochastic Constraints for Fast Image Correspondence Search with Uncertain Terrain Model. IEEE Transactions on Aerospace Electronic Systems, 42(3):973-982, July 2006.

[32] Greg Welch, Gary Bishop. An Introduction to the Kalman Filter. SIGGRAPH 2001, Course 8, 2001.